Curso Inteligência Artificial: do Zero ao Infinito

Introducão do modelo Perceptron

Universidade Federal de Mato Grosso

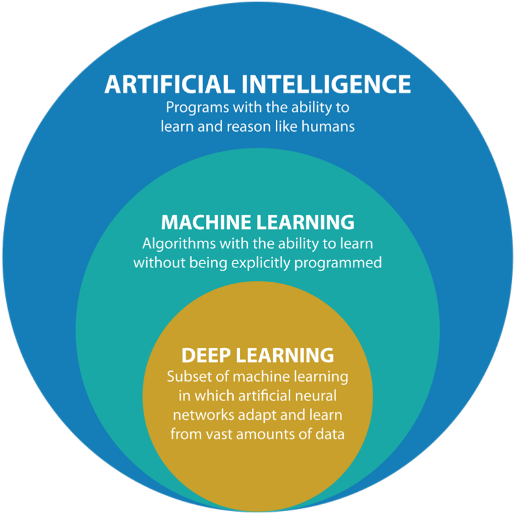
Agenda

Objetivos

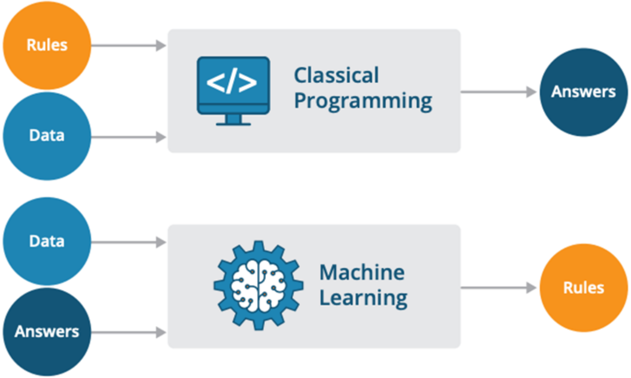
Na aula de hoje, nosso objetivo é:

- Apresentar o modelo Perceptron.
- Compreender e implementar o Neurônio Artificial.

Redes Neurais



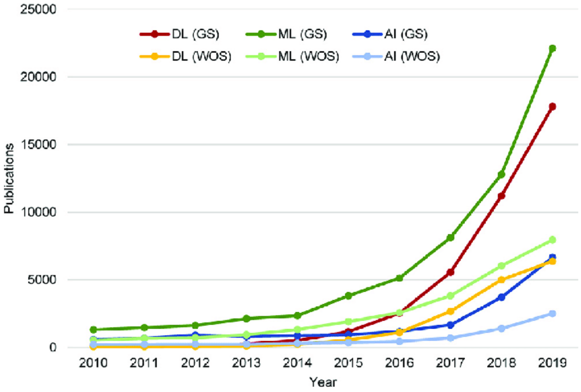
Redes Neurais



Fonte: Traditional Programming vs Machine Learning.

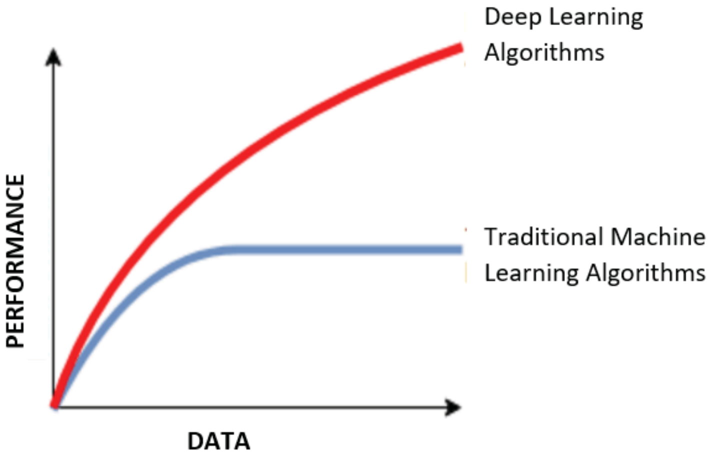
Redes Neurais

Crescimento DL x ML x AI



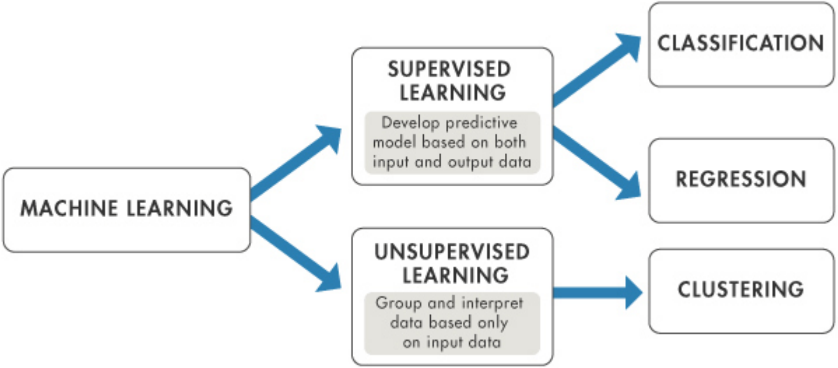
Número de publicações sobre deep learning (DL), machine learning (ML), ou artificial intelligence (AI), de acordo com Google Scholar (GS) e Web of Science (WOS). Source: A Bird's-Eye View of Deep Learning in Bioimage Analysis

Redes Neurais



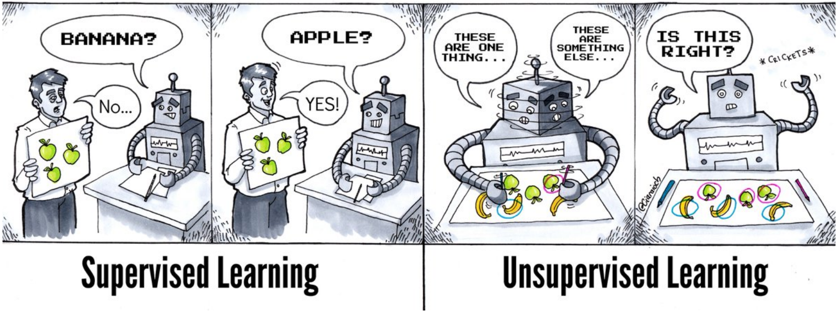
Source: Offline Arabic Handwriting Recognition Using Deep Learning: Comparative Study

Redes Neurais
Supervised x Unsupervised



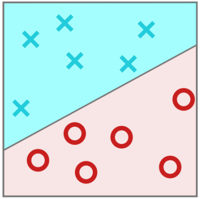
Redes Neurais

Supervised x Unsupervised

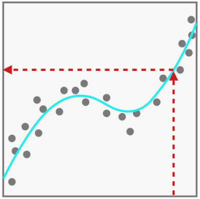


Redes Neurais

Regression x Classification



Classification



Regression

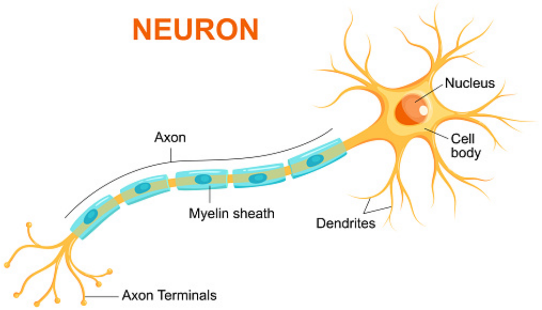
Rede Neural Artificial

- Rede Neural Artificial (RNA) é uma abstração da rede neural biológica presente no cérebro humano.
- A versão artificial é inspirada na forma como as redes neurais biológicas processam informações.
- O primeiro modelo foi apresentado em 1943, por Warren McCulloch e Walter Pitts.

Referência: Artigo McCulloch e Pitts

Neurônio Biológico

- Cada neurônio possui um corpo central, diversos dendritos e um axônio.
- Os dendritos recebem sinais elétricos de outros neurônios através das sinapses.
- O corpo celular processa a informação e envia para outro neurônio.



Perceptron

- O Perceptron é um algoritmo clássico para o modelo neural de aprendizagem.
- Foi apresentado em 1958 por Frank Rosenblatt.
- É incrivelmente simples e funciona surpreendentemente bem.

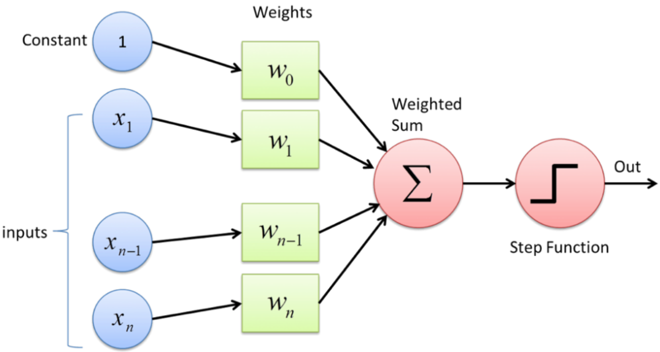
Referência: Livro Frank Rosenblatt

Perceptron

- É formado por um único neurônio que recebe entradas numéricas.
  - n entradas, 1 saída
- Pondera cada entrada por um peso (ou sinapse).
  - Realiza um somatório das entradas ponderadas.
- Verifica se a soma atinge um limiar.
  - Utiliza função de ativação denominada **Step**.

Perceptron

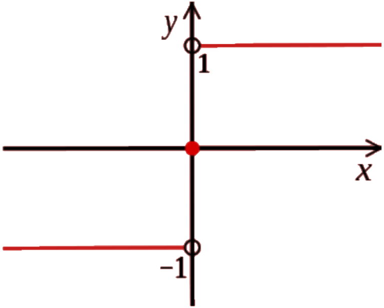
Uma RNA é um componente que calcula a soma ponderada de vários *inputs*, aplica uma função e passa o resultado adiante.



Perceptron

Step Function

$$Step(x) = \begin{cases} 1, & \text{se } x \geq 0. \\ 0, & \text{caso contrário.} \end{cases}$$



sign(x)

Perceptron

A saída do Perceptron é definida como uma combinação linear da entrada x e dos pesos w.

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

O valor de z é dado por:

$$z = w_1x_1 + w_2x_2 + ... + x_mx_m$$

Perceptron

É necessário incluir um valor w_0, chamado *bias*, que será multiplicado por $x_0 = 1$.

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m$$

que pode ser escrito da seguinte forma:

$$z = \sum_{i=0}^{m} w_i x_i$$

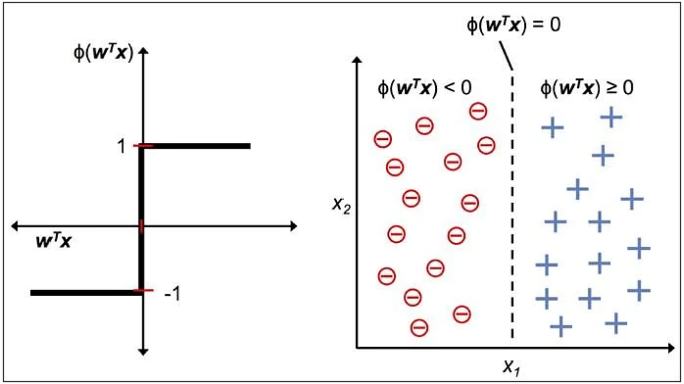
Perceptron

O valor final é obtido a partir da função *step*.

$$\phi(x) = \begin{cases} 1, & \text{se } x \geq 0. \\ 0, & \text{caso contrário.} \end{cases}$$

Perceptron

O resultado da função *step* é usado para separar as classes de um problema.



Perceptron

Na notação matricial, fica da seguinte forma:

$$\hat{y} = \phi(W^T x + b)$$

em que:

- $w \in R^N$ é um parâmetro, denominado **pesos** do modelo.
- **b** é o bias do modelo.
- ϕ é denominada **função de ativação**, em que no perceptron é utilizada a função *step*.

Perceptron

Loss

O *erro*, denominado **loss**, é calculado da seguinte forma:

$$e = y - \hat{y}$$

O objetivo do perceptron é obter erro igual a zero.

y	$\hat{y}$	Loss
+1	+1	0
+1	-1	2
-1	-1	0
-1	+1	-2

- O aprendizado do perceptron é feito atualizando os pesos baseado na *loss*.
- Para cada amostra classificada errôneamente, o vetor de pesos é atualizado da seguinte forma:

$$w_{t+1} = w_t - \alpha Loss(\hat{y}, y)$$

- O parâmetro α é denominada **taxa de aprendizagem** (em inglês *learning rate*), e define em quanto os pesos serão corrigidos.

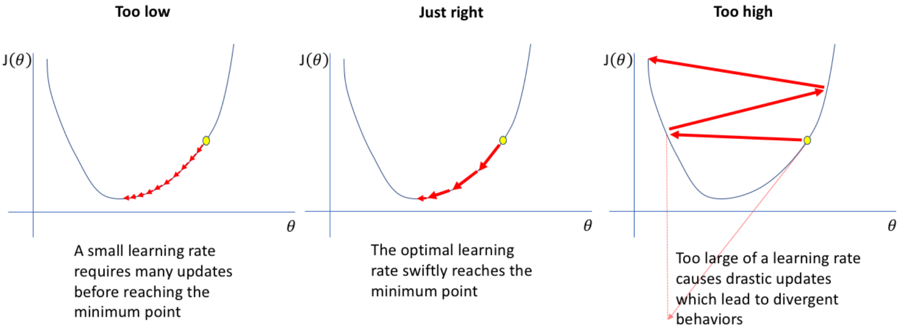
Perceptron

Learning Rate

- A **taxa de aprendizado** α tem grande influência durante o processo de treinamento da rede neural.
- Um valor muito baixo torna o aprendizado da rede muito lento
- Ao passo que uma valor muito alto provoca oscilações no treinamento e impede a convergência do processo de aprendizado.

Perceptron

Learning Rate



Perceptron

Learning Rate

- Todo o processo de aprendizagem é repetido, até que o modelo tenha convergido.
- O total de repetições é denominado **número de épocas** e também é um parâmetro definido pelo usuário.

Perceptron

Algoritmo 1: Treinamento Perceptron

Entrada: $X = \{(x_1, ...x_m)\}$ $y = \{y_1, ...y_m\}\}$

Saída: W

início

    `// Inicializa` W `e` b `com valores aleatórios`

    $W = [w_1, ..., w_m]$

    $b = [w_0]$

    repita

        para *cada amostra* $x_i \in X$ **faça**

            $\hat{y}_i \leftarrow \phi(w_i x_i + b)$

            $\mathrm{L} \leftarrow (y_i - \hat{y}_i)$

            $w_i \leftarrow w_i - \alpha L$

            $b \leftarrow b - \alpha L$

    até *n épocas*;

Implementação em Python

Implementação em Python utilizando Numpy

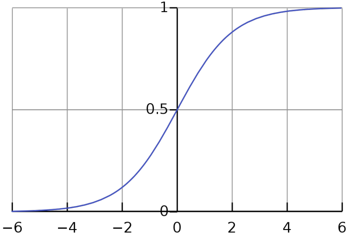
Neurônio Artificial

- Composta por apenas um neurônio:
  - N entradas, 1 saída
  - Função de ativação: *sigmoid*
  - Função de custo: entropia cruzada (*cross-entropy*)
- Por definição, é um classificador binário
- Introduz não-linearidade ao Perceptron
- Pequenas mudanças nos parâmetros, causam pequenas mudanças na saída.
- Não é utilizado para fazer regressão linear
- Utiliza o Gradiente Descendente.

Neurônio Artificial

Sigmoid Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



Neurônio Artificial

Binary Cross Entropy

Função Custo:

$$J(z) = -1\frac{1}{N}\sum_{i}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

Neurônio Artificial

Binary Cross Entropy

$$J(z) = -1\frac{1}{N}\sum_{i}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

y	$\hat{y}$	J
0	0	
0	1	
1	0	
1	1	

Neurônio Artificial

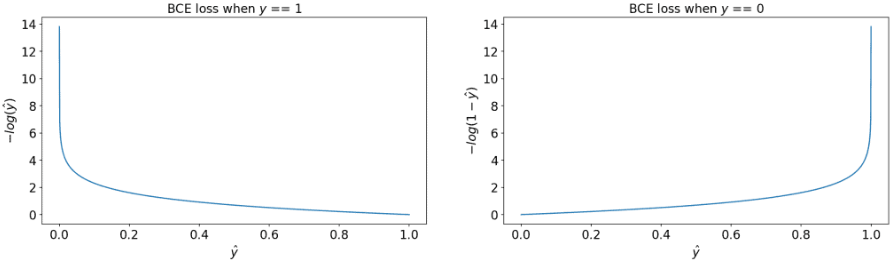
Binary Cross Entropy

$$J(z) = -1\frac{1}{N}\sum_{i}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

y	$\hat{y}$	J
0	0	0
0	1	∞
1	0	∞
1	1	0

Neurônio Artificial

Logistic Function



Para calcular a derivada da Função de Custo é necessário substituir a expressão:

$$\log(\hat{y}) = \log \frac{1}{1+e^{-z}} = \log(1) - log(1+e^{-z}) = -\log(1+e^{-z})$$

Com isso, a derivada da função de custo $J(z)$ é:

$$\frac{\partial}{\partial w} J(z) = \sum (y_i - \hat{y}_i) x_i$$

Neurônio Artificial

Algoritmo 2: Treinamento Neurônio Artificial

Entrada: $X = \{(x_1, ...x_m)\}\ y = \{y_1, ...y_m\}\}$

Saída: W

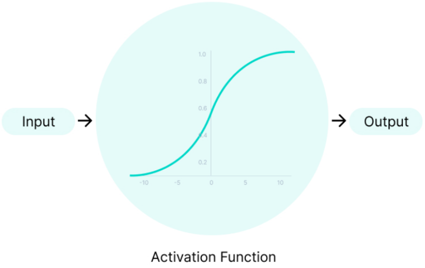
início

    // Inicializa W e b com valores aleatórios
    $W = [w_1, ..., w_m]$
    $b = [w_0]$
    repita
        para *cada amostra* $x_i \in X$ **faça**
            $\hat{y}_i \leftarrow sigmoid(w_i x_i + b)$
            $J \leftarrow (y_i - \hat{y}_i)$
            $w_i \leftarrow w_i - \alpha \frac{\partial J}{\partial w}$
            $b \leftarrow b - \alpha \frac{\partial J}{\partial w}$
    até *n épocas*;

Funções de Ativação

Funções de Ativação

- O propósito de se utilizar uma função de ativação é adicionar **não-linearidade** à NN.
- Introduzem uma etapa adicional no processo de *propagação*.
  - Mas seu uso vale a pena!
- A escolha da função de ativação tem grande impacto na capacidade e performance da NN.
  - Diferentes funções de ativações podem ser utilizadas em diferentes partes do modelo.

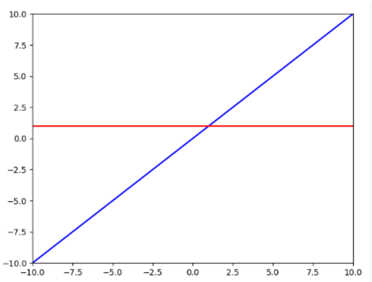


Funções de Ativação
Linear

Função de Ativação **Linear**:

- Chamada de *função identidade*.
- Varia de $-\infty$ a $+\infty$.
- Utilizada na camada de saída de um problema de **regressão**.
- Desvantagens:
  - Não é possível utilizar *backpropagation*, pois a derivada é uma constante.
  - Várias camadas utilizando função linear podem ser substituídas por uma única camada.

$$y = f(x)$$



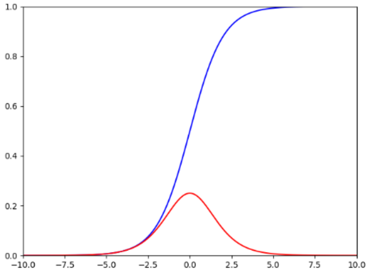
$$\frac{\partial y}{\partial x} = 1$$

Funções de Ativação
Sigmoid

Função de Ativação **Sigmoid**:

- Conhecida como *função logística*
- Utilizada para *regressão logística* (classificação)
- Varia de 0 a 1.
- A saída é interpretada como a probabilidade de uma amostra pertencer a determinada classe
- Desvantagens:
  - Ocorre dissipação do gradiente.
  - Convergência lenta
  - Saída não é centrada em zero.

$$y = \frac{1}{1 + e^{-x}}$$

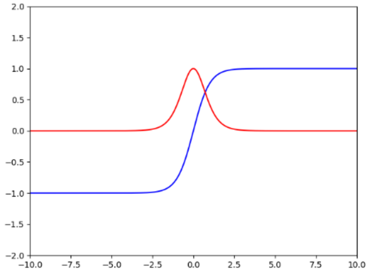


$$\frac{\partial y}{\partial x} = y(1 - y)$$

Função de Ativação **Tanh**:

- Significantemente melhor do que a Sigmoide.
- Varia de -1 a 1.
- A saída é centrada em zero.
- Ocorre uma menor dissipação do gradiente.

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



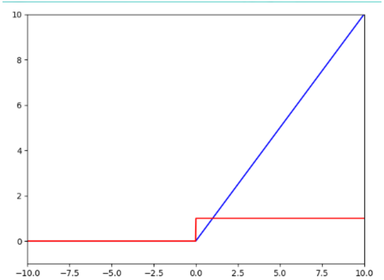
$$\frac{\partial y}{\partial x} = 1 - y^2$$

Funções de Ativação
ReLU

Função de Ativação **ReLU**:

- Função de atviação mais usada.
- Varia de 0 a $+\infty$.
- Vantagens:
  - Simples e eficiente.
  - Evita o problema de dissipação do gradiente.
  - Melhora a convergência
- Desvantagens:
  - Utilizada nas camadas ocultas.
  - Pode matar alguns neurônios.

$$y = max(0, x)$$



$$\frac{\partial y}{\partial x} = \begin{cases} 0, & \text{se } x \leq 0. \\ 1, & \text{se } x > 0. \end{cases}$$

Funções de Ativação

Leaky ReLU

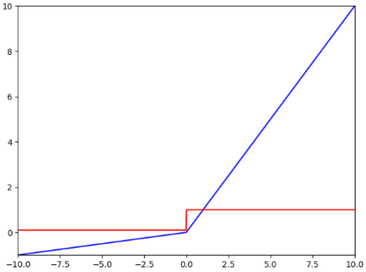
Função de Ativação **Leaky ReLU**:

- Pequena variação da ReLU.
- Varia de $-\infty$ a $+\infty$.

$$y = \begin{cases} \alpha x, & \text{se } x \leq 0. \\ x, & \text{se } x > 0. \end{cases}$$

- Vantagens:
  - Reduz a possibilidade de "matar" neurônios.
- Desvantagens:
  - Deve ser utilizada apenas nas camadas ocultas.

$$y = max(0, x)$$



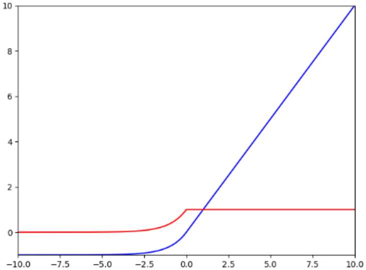
$$\frac{\partial y}{\partial x} = \begin{cases} \alpha, & \text{se } x \leq 0. \\ 1, & \text{se } x > 0. \end{cases}$$

Funções de Ativação
eLU

Função de Ativação **eLU**:

- É uma pequena variação da ReLU.
- Varia de $-\infty$ a $+\infty$.

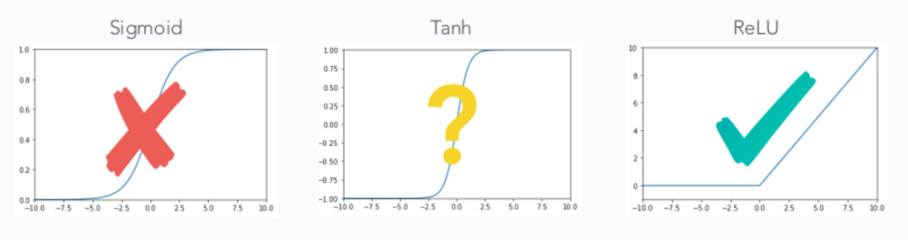
$$y = \begin{cases} \alpha(e^x - 1) & \text{se } x \leq 0. \\ x, & \text{se } x > 0. \end{cases}$$



- Vantagens:
  - Reduz a possibilidade de “matar"neurônios.
- Desvantagens:
  - Deve ser utilizada apenas nas camadas ocultas.

$$\frac{\partial y}{\partial x} = \begin{cases} y + \alpha, & \text{se } x \leq 0. \\ 1, & \text{se } x > 0. \end{cases}$$

Funções de Ativação



Funções de Ativação

Referências

- How to Choose an Activation Function for Deep Learning
- Activation Functions in Neural Networks

Funções de Custo

Funções Custo

- *Loss* se aplica para um único elemento do conjunto de treinamento.
- Custo se refere a *Loss* calculada sobre todo o conjunto de treinamento (*mini-batch*).

Funções Custo

MAE

mean of absolute errors

$$J = \frac{1}{N}\sum |y_i - \hat{y}_i|$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{1}{N}\begin{cases} +1 & se\ \hat{y} > y \\ -1 & se\ \hat{y} < y \end{cases}$$

MSE

mean of squared errors

$$J = \frac{1}{2N}\sum (y_i - \hat{y}_i)^2$$

$$\frac{\partial J}{\partial \hat{y}} = -(y - \hat{y})\frac{1}{N}$$

Funções Custo

Binary Cross Entropy

$$J(\hat{y}, y) = \frac{-1}{m} \sum_{i=1}^{m} y_i \log(\hat{y}_i) + (1 - y_i)(\log(1 - \hat{y}))$$

Funções Custo

Binary Cross Entropy

$$
\begin{aligned}
J(\hat{y}, y) &= \frac{-1}{m} \sum_{i=1}^{m} y_i \log(\hat{y}_i) + (1 - y_i)(\log(1 - \hat{y})) \\
\frac{\partial J(\hat{y}, y)}{\partial w} &= ?
\end{aligned}
$$

Funções Custo

Binary Cross Entropy

Usando a regra da cadeia:

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J(w)}{\partial z}\frac{\partial z}{\partial h}\frac{\partial h}{\partial w}$$

Referências

- Deep Learning Book - Perceptron
- How To Implement The Perceptron Algorithm From Scratch In Python

Curso Inteligência Artificial: do Zero ao Infinito

Introducão do modelo Perceptron

Universidade Federal de Mato Grosso